Apache Spark - Practice set 4

baburam@fusemachines.com Switch account

* Indicates required question

Email *

Record baburam@fusemachines.com as the email to be included with my response

Which of the following statements about Spark's execution hierarchy is correct? 1 point

- In Spark's execution hierarchy, a job may reach over multiple stage boundaries.
- In Spark's execution hierarchy, manifests are one layer above jobs.
- In Spark's execution hierarchy, a stage comprises multiple jobs.
- In Spark's execution hierarchy, executors are the smallest unit.
- In Spark's execution hierarchy, tasks are one layer above slots.

Which of the following describes slots? 1 point

- Slots are dynamically created and destroyed in accordance with an executor's workload.
- To optimize I/O performance, Spark stores data on disk in multiple slots.
- A Java Virtual Machine (JVM) working as an executor can be considered as a pool of slots for task execution.
- A slot is always limited to a single core.
- Slots are the communication interface for executors and are used for receiving commands and sending results to the driver.

Which of the following describes the conversion of a computational query into an execution plan in Spark? 1 point

- Spark uses the catalog to resolve the optimized logical plan.
- The catalog assigns specific resources to the optimized memory plan.
- The executed physical plan depends on a cost optimization from a previous stage.
- Depending on whether DataFrame API or SQL API are used, the physical plan may differ.
- The catalog assigns specific resources to the physical plan.

Which of the following describes characteristics of the Spark driver? 1 point

- The Spark driver requests the transformation of operations into DAG computations from the worker nodes.
- If set in the Spark configuration, Spark scales the Spark driver horizontally to improve parallel processing performance.
- The Spark driver processes partitions in an optimized, distributed fashion.
- In a non-interactive Spark application, the Spark driver automatically creates the SparkSession object.
- The Spark driver's responsibility includes scheduling queries for execution on worker nodes.

Which of the following describes executors? 1 point

- Executors host the Spark driver on a worker-node basis.
- Executors are responsible for carrying out work that they get assigned by the driver.
- After the start of the Spark application, executors are launched on a per-task basis.
- Executors are located in slots inside worker nodes.
- The executors' storage is ephemeral and as such it defers the task of caching data directly to the worker node thread.

Which of the following statements about DAGs is correct? 1 point

- DAGs help direct how Spark executors process tasks, but are a limitation to the proper execution of a query when an executor fails.
- DAG stands for "Directing Acyclic Graph".
- Spark strategically hides DAGs from developers, since the high degree of automation in Spark means that developers never need to consider DAG layouts.
- In contrast to transformations, DAGs are never lazily executed.
- DAGs can be decomposed into tasks that are executed in parallel.

Which of the following DataFrame methods is classified as a transformation? 1 point

- DataFrame.count()
- DataFrame.show()
- DataFrame.select()
- DataFrame.foreach()
- DataFrame.first()

Which of the following statements about lazy evaluation is incorrect? 1 point

- Predicate pushdown is a feature resulting from lazy evaluation.
- Execution is triggered by transformations.
- Spark will fail a job only during execution, but not during definition.
- Accumulators do not change the lazy evaluation model of Spark.
- Lineages allow Spark to coalesce transformations into stages.

Which of the following describes how Spark achieves fault tolerance? 1 point

- Spark helps fast recovery of data in case of a worker fault by providing the MEMORY_AND_DISK storage level option.
- If an executor on a worker node fails while calculating an RDD, that RDD can be recomputed by another executor using the lineage.
- Spark builds a fault-tolerant layer on top of the legacy RDD data system, which by itself is not fault tolerant.
- Due to the mutability of DataFrames after transformations, Spark reproduces them using observed lineage in case of worker node failure.
- Spark is only fault-tolerant if this feature is specifically enabled via the spark.fault_recovery.enabled property.

Which is the highest level in Spark's execution hierarchy? 1 point

- Task
- Executor
- Slot
- Job
- Stage

Which of the following describes Spark's way of managing memory? 1 point

- Spark uses a subset of the reserved system memory.
- Storage memory is used for caching partitions derived from DataFrames.
- As a general rule for garbage collection, Spark performs better on many small objects than few big objects.
- Disabling serialization potentially greatly reduces the memory footprint of a Spark application.
- Spark's memory usage can be divided into three categories: Execution, transaction, and storage.

Which of the following statements about Spark's DataFrames is incorrect? 1 point

- Spark's DataFrames are immutable.
- Spark's DataFrames are equal to Python's DataFrames.
- Data in DataFrames is organized into named columns.
- RDDs are at the core of DataFrames.
- The data in DataFrames may be split into multiple chunks.

Which of the following statements about Spark's configuration properties is incorrect? 1 point

- The maximum number of tasks that an executor can process at the same time is controlled by the spark.task.cpus property.
- The maximum number of tasks that an executor can process at the same time is controlled by the spark.executor.cores property.
- The default value for spark.sql.autoBroadcastJoinThreshold is 10MB.
- The default number of partitions to use when shuffling data for joins or aggregations is 300.
- The default number of partitions returned from certain transformations can be controlled by the spark.default.parallelism property.

Which of the following describes a way for resizing a DataFrame from 16 to 8 partitions in the most efficient way? 1 point

- Use operation DataFrame.repartition(8) to shuffle the DataFrame and reduce the number of partitions.
- Use operation DataFrame.coalesce(8) to fully shuffle the DataFrame and reduce the number of partitions.
- Use a narrow transformation to reduce the number of partitions.
- Use a wide transformation to reduce the number of partitions.
- Use operation DataFrame.coalesce(0.5) to halve the number of partitions in the DataFrame.

Which of the following DataFrame operators is never classified as a wide transformation? 1 point

- DataFrame.sort()
- DataFrame.aggregate()
- DataFrame.repartition()
- DataFrame.select()
- DataFrame.join()

Which of the following statements about data skew is incorrect? 1 point

- Spark will not automatically optimize skew joins by default.
- Broadcast joins are a viable way to increase join performance for skewed data over sort-merge joins.
- In skewed DataFrames, the largest and the smallest partition consume very different amounts of memory.
- To mitigate skew, Spark automatically disregards null values in keys when joining.
- Salting can resolve data skew.

Which of the following describes the characteristics of accumulators? 1 point

- Accumulators are used to pass around lookup tables across the cluster.
- All accumulators used in a Spark application are listed in the Spark UI.
- Accumulators can be instantiated directly via the accumulator(n) method of the pyspark.RDD module.
- Accumulators are immutable.
- If an action including an accumulator fails during execution and Spark manages to restart the action and complete it successfully, only the successful attempt will be counted in the accumulator.

Which of the following code blocks returns a DataFrame with a single column in which all items in column attributes of DataFrame itemsDf are listed that contain the letter i? 1 point

Sample of DataFrame `itemsDf`:

```
+------+----------------------------------+-----------------------------+--------
-----------+
|itemId|itemName                          |attributes
|supplier           |
+------+----------------------------------+-----------------------------+--------
-----------+
|1     |Thick Coat for Walking in the Snow|[blue, winter, cozy]         |Sports
Company Inc.|
|2     |Elegant Outdoors Summer Dress     |[red, summer, fresh, cooling]|YetiX
            |
|3     |Outdoors Backpack                 |[green, summer, travel]      |Sports
Company Inc.|
+------+----------------------------------+-----------------------------+--------
-----------+
```

- itemsDf.select(explode("attributes").alias("attributes_exploded")).filter(attributes_exploded.contains("i"))
- itemsDf.explode(attributes).alias("attributes_exploded").filter(col("attributes_exploded").contains("i"))
- itemsDf.select(explode("attributes")).filter("attributes_exploded".contains("i"))
- itemsDf.select(explode("attributes").alias("attributes_exploded")).filter(col("attributes_exploded").contains("i"))
- itemsDf.select(col("attributes").explode().alias("attributes_exploded")).filter(col("attributes_exploded").contains("i"))

Which of the following code blocks returns all unique values of column storeId in DataFrame transactionsDf? 1 point

- transactionsDf["storeId"].distinct()
- transactionsDf.select("storeId").distinct()
- transactionsDf.filter("storeId").distinct()
- transactionsDf.select(col("storeId").distinct())
- transactionsDf.distinct("storeId")

Which of the following code blocks stores a part of the data in DataFrame itemsDf on executors? 1 point

- itemsDf.cache().count()
- itemsDf.cache(eager=True)
- cache(itemsDf)
- itemsDf.cache().filter()
- itemsDf.rdd.storeCopy()

Which of the following code blocks selects all rows from DataFrame transactionsDf in which column productId is zero or smaller or equal to 3? 1 point

- transactionsDf.filter(productId==3 or productId<1)
- transactionsDf.filter((col("productId")==3) or (col("productId")<1))
- transactionsDf.filter(col("productId")==3 | col("productId")<1)
- transactionsDf.where("productId"=3).or("productId"<1))
- transactionsDf.filter((col("productId")==3) | (col("productId")<1))

Which of the following code blocks sorts DataFrame transactionsDf both by column storeId in ascending and by column productId in descending order, in this priority? 1 point

- transactionsDf.sort("storeId", asc("productId"))
- transactionsDf.sort(col(storeId)).desc(col(productId))
- transactionsDf.order_by(col(storeId), desc(col(productId)))
- transactionsDf.sort("storeId", desc("productId"))
- transactionsDf.sort("storeId").sort(desc("productId"))

The code block displayed below contains an error. The code block should produce a DataFrame with color as the only column and three rows with color values of red, blue, and green, respectively. Find the error. Code block:spark.createDataFrame([("red",), ("blue",), ("green",)], "color") 1 point

- Instead of calling spark.createDataFrame, just DataFrame should be called.
- The commas in the tuples with the colors should be eliminated.
- The colors red, blue, and green should be expressed as a simple Python list, and not a list of tuples.
- Instead of color, a data type should be specified.
- The "color" expression needs to be wrapped in brackets, so it reads ["color"].

The code block displayed below contains an error. The code block should return all rows of DataFrame transactionsDf, but including only columns storeId and predError. Find the error. Code block:spark.collect(transactionsDf.select("storeId", "predError")) 1 point

- Instead of select, DataFrame transactionsDf needs to be filtered using the filter operator.
- Columns storeId and predError need to be represented as a Python list, so they need to be wrapped in brackets ([]).
- The take method should be used instead of the collect method.
- Instead of collect, collectAsRows needs to be called.
- The collect method is not a method of the SparkSession object.

The code block shown below should store DataFrame transactionsDf on two different executors, utilizing the executors' memory as much as possible, but not writing anything to disk. Choose the answer that correctly fills the blanks in the code block to accomplish this. 1. from pyspark import StorageLevel 2.transactionsDf.__1__(StorageLevel.__2__).__3__ 1 point

- 1. Cache 2. MEMORY_ONLY_2 3. count()
- 1. Persist 2. DISK_ONLY_2 3. count()
- 1. Persist 2. MEMORY_ONLY_2 3. select()
- 1. Cache 2. DISK_ONLY_2 3. count()
- 1. Persist 2. MEMORY_ONLY_2 3. count()

The code block displayed below contains an error. The code block should return a copy of DataFrame transactionsDf where the name of column transactionId has been changed to transactionNumber. Find the error. Code block:transactionsDf.withColumn("transactionNumber", "transactionId") 1 point

- The arguments to the withColumn method need to be reordered.
- The arguments to the withColumn method need to be reordered and the copy() operator should be appended to the code block to ensure a copy is returned.
- The copy() operator should be appended to the code block to ensure a copy is returned.
- Each column name needs to be wrapped in the col() method and method withColumn should be replaced by method withColumnRenamed.
- The method withColumn should be replaced by method withColumnRenamed and the arguments to the method need to be reordered.

Which of the following code blocks performs an inner join between DataFrame itemsDf and DataFrame transactionsDf, using columns itemId and transactionId as join keys, respectively? 1 point

- itemsDf.join(transactionsDf, "inner", itemsDf.itemId == transactionsDf.transactionId)
- itemsDf.join(transactionsDf, itemId == transactionId)
- itemsDf.join(transactionsDf, itemsDf.itemId == transactionsDf.transactionId, "inner")
- itemsDf.join(transactionsDf, "itemsDf.itemId == transactionsDf.transactionId", "inner")
- itemsDf.join(transactionsDf, col(itemsDf.itemId) == col(transactionsDf.transactionId))

The code block shown below should return only the average prediction error (column predError) of a random subset, without replacement, of approximately 15% of rows in DataFrame transactionsDf. Choose the answer that correctly fills the blanks in the code block to accomplish this.transactionsDf.__1__(__2__, __3__).__4__(avg('predError')) 1 point

- 1. Sample 2. True 3. 0.15 4. filter
- 1. Sample 2. False 3. 0.15 4. select
- 1. Sample 2. 0.85 3. False 4. select
- 1. Fraction 2. 0.15 3. True 4. where
- 1. Fraction 2. False 3. 0.85 4. select

Which of the following code blocks returns a single-column DataFrame showing the number of words in column supplier of DataFrame itemsDf? 1 point

Sample of DataFrame `itemsDf`:

```
+------+-----------------------------+-------------------+
|itemId|attributes                   |supplier           |
+------+-----------------------------+-------------------+
|1     |[blue, winter, cozy]         |Sports Company Inc.|
|2     |[red, summer, fresh, cooling]|YetiX              |
|3     |[green, summer, travel]      |Sports Company Inc.|
+------+-----------------------------+-------------------+
```

- itemsDf.split("supplier", " ").count()
- itemsDf.split("supplier", " ").size()
- itemsDf.select(word_count("supplier"))
- spark.select(size(split(col(supplier), " ")))
- itemsDf.select(size(split("supplier", " ")))

Which of the following code blocks stores DataFrame itemsDf in executor memory and, if insufficient memory is available, serializes it and saves it to disk? 1 point

- itemsDf.persist(StorageLevel.MEMORY_ONLY)
- itemsDf.cache(StorageLevel.MEMORY_AND_DISK)
- itemsDf.store()
- itemsDf.cache()
- itemsDf.write.option('destination', 'memory').save()

Which of the following code blocks adds a column predErrorSqrt to DataFrame transactionsDf that is the square root of column predError? 1 point

- transactionsDf.withColumn("predErrorSqrt", sqrt(predError))
- transactionsDf.select(sqrt(predError))
- transactionsDf.withColumn("predErrorSqrt", col("predError").sqrt())
- transactionsDf.withColumn("predErrorSqrt", sqrt(col("predError")))
- transactionsDf.select(sqrt("predError"))

Which of the following code blocks reorders the values inside the arrays in column attributes of DataFrame itemsDf from last to first one in the alphabet? 1 point

```
+------+-----------------------------+-------------------+
|itemId|attributes                   |supplier           |
+------+-----------------------------+-------------------+
|1     |[blue, winter, cozy]         |Sports Company Inc.|
|2     |[red, summer, fresh, cooling]|YetiX              |
|3     |[green, summer, travel]      |Sports Company Inc.|
+------+-----------------------------+-------------------+
```

- itemsDf.withColumn('attributes', sort_array(col('attributes').desc()))
- itemsDf.withColumn('attributes', sort_array(desc('attributes')))
- itemsDf.withColumn('attributes', sort(col('attributes'), asc=False))
- itemsDf.withColumn("attributes", sort_array("attributes", asc=False))
- itemsDf.select(sort_array("attributes"))

Which of the following code blocks returns a copy of DataFrame transactionsDf where the column storeId has been converted to string type? 1 point

- transactionsDf.withColumn("storeId", convert("storeId", "string"))
- transactionsDf.withColumn("storeId", col("storeId", "string"))
- transactionsDf.withColumn("storeId", col("storeId").convert("string"))
- transactionsDf.withColumn("storeId", col("storeId").cast("string"))
- transactionsDf.withColumn("storeId", convert("storeId").as("string"))

Which of the following code blocks applies the boolean-returning Python function evaluateTestSuccess to column storeId of DataFrame transactionsDf as a user-defined function? 1 point

- 1|from pyspark.sql import types as T 2|evaluateTestSuccessUDF = udf(evaluateTestSuccess, T.BooleanType()) 3|transactionsDf.withColumn("result", evaluateTestSuccessUDF(col("storeId")))
- 1|evaluateTestSuccessUDF = udf(evaluateTestSuccess) 2|transactionsDf.withColumn("result", evaluateTestSuccessUDF(storeId))
- 1|from pyspark.sql import types as T 2|evaluateTestSuccessUDF = udf(evaluateTestSuccess, T.IntegerType()) 3|transactionsDf.withColumn("result", evaluateTestSuccess(col("storeId")))
- 1| evaluateTestSuccessUDF = udf(evaluateTestSuccess) 2| transactionsDf.withColumn("result", evaluateTestSuccessUDF(col("storeId")))
- 1| from pyspark.sql import types as T 2| evaluateTestSuccessUDF = udf(evaluateTestSuccess, T.BooleanType()) 3| transactionsDf.withColumn("result", evaluateTestSuccess(col("storeId")))

Which of the following code blocks returns a copy of DataFrame transactionsDf in which column productId has been renamed to productNumber? 1 point

- transactionsDf.withColumnRenamed("productId", "productNumber")
- transactionsDf.withColumn("productId", "productNumber")
- transactionsDf.withColumnRenamed("productNumber", "productId")
- transactionsDf.withColumnRenamed(col(productId), col(productNumber))
- transactionsDf.withColumnRenamed(productId, productNumber)

Which of the following code blocks returns a copy of DataFrame transactionsDf that only includes columns transactionId, storeId, productId and f?

1 point

Sample of DataFrame `transactionsDf`:

```
+-------------+---------+-----+-------+---------+----+
|transactionId|predError|value|storeId|productId|   f|
+-------------+---------+-----+-------+---------+----+
|            1|        3|    4|     25|        1|null|
|            2|        6|    7|      2|        2|null|
|            3|        3| null|     25|        3|null|
+-------------+---------+-----+-------+---------+----+
```

- transactionsDf.drop(col("value"), col("predError"))
- transactionsDf.drop("predError", "value")
- transactionsDf.drop(value, predError)
- transactionsDf.drop(["predError", "value"])
- transactionsDf.drop([col("predError"), col("value")])

The code block shown below should return a one-column DataFrame where the column storeId is converted to string type. Choose the answer that correctly fills the blanks in the code block to accomplish this.transactionsDf.__1__(__2__.__3__(__4__))

1 point

- 1. Select 2. col("storeId") 3. Cast 4. StringType
- 1. Select 2. col("storeId") 3. As 4. StringType
- 1. Cast 2. "storeId" 3. As 4. StringType()
- 1. Select 2. col("storeId") 3. Cast 4. StringType()
- 1. Select 2. storeId 3. Cast 4. StringType()

Which of the following code blocks creates a new one-column, two-row DataFrame dfDates with column date of type timestamp? 1 point

Option 13

Option 2

Option 3

Option 4

Option 5

The code block displayed below contains an error. The code block should save DataFrame transactionsDf at path path as a parquet file, appending to any existing parquet file. Find the error. Code block:transactionsDf.format("parquet").option("mode", "append").save(path) 1 point

- The code block is missing a reference to the DataFrameWriter.
- save() is evaluated lazily and needs to be followed by an action.
- The mode option should be omitted so that the command uses the default mode.
- The code block is missing a bucketBy command that takes care of partitions.
- Given that the DataFrame should be saved as parquet file, path is being passed to the wrong method.

Which of the following code blocks returns a single row from DataFrame transactionsDf? 1 point

Full DataFrame `transactionsDf`:

```
+-------------+---------+-----+-------+---------+----+
|transactionId|predError|value|storeId|productId|   f|
+-------------+---------+-----+-------+---------+----+
|            1|        3|    4|     25|        1|null|
|            2|        6|    7|      2|        2|null|
|            3|        3| null|     25|        3|null|
|            4|     null| null|      3|        2|null|
|            5|     null| null|   null|        2|null|
|            6|        3|    2|     25|        2|null|
+-------------+---------+-----+-------+---------+----+
```

- transactionsDf.where(col("storeId").between(3,25))
- transactionsDf.filter((col("storeId")!=25) | (col("productId")==2))
- transactionsDf.filter(col("storeId")==25).select("predError","storeId").distinct()
- transactionsDf.select("productId", "storeId").where("storeId == 2 OR storeId != 25")
- transactionsDf.where(col("value").isNull()).select("productId", "storeId").distinct()

Which of the following code blocks returns approximately 1000 rows, some of them potentially being duplicates, from the 2000-row DataFrame transactionsDf that only has unique rows? 1 point

- transactionsDf.sample(True, 0.5)
- transactionsDf.take(1000).distinct()
- transactionsDf.sample(False, 0.5)
- transactionsDf.take(1000)
- transactionsDf.sample(True, 0.5, force=True)

Which of the following code blocks returns a DataFrame showing the mean value of column "value" of DataFrame transactionsDf, grouped by its column storeId? 1 point

- transactionsDf.groupBy(col(storeId).avg())
- transactionsDf.groupBy("storeId").avg(col("value"))
- transactionsDf.groupBy("storeId").agg(avg("value"))
- transactionsDf.groupBy("storeId").agg(average("value"))
- transactionsDf.groupBy("value").average()

Which of the following code blocks concatenates rows of DataFrames transactionsDf and transactionsNewDf, omitting any duplicates? 1 point

- transactionsDf.concat(transactionsNewDf).unique()
- transactionsDf.union(transactionsNewDf).distinct()
- spark.union(transactionsDf, transactionsNewDf).distinct()
- transactionsDf.join(transactionsNewDf, how="union").distinct()
- transactionsDf.union(transactionsNewDf).unique()

In which order should the code blocks shown below be run in order to assign articlesDf a DataFrame that lists all items in column attributes ordered by the number of times these items occur, from most to least often? 1 point

Sample of DataFrame `articlesDf`:

```
+------+-----------------------------+-------------------+
|itemId|attributes                   |supplier           |
+------+-----------------------------+-------------------+
|1     |[blue, winter, cozy]         |Sports Company Inc.|
|2     |[red, summer, fresh, cooling]|YetiX              |
|3     |[green, summer, travel]      |Sports Company Inc.|
+------+-----------------------------+-------------------+
```

1. `articlesDf = articlesDf.groupby("col")`
2. `articlesDf = articlesDf.select(explode(col("attributes")))`
3. `articlesDf = articlesDf.orderBy("count").select("col")`
4. `articlesDf = articlesDf.sort("count",ascending=False).select("col")`
5. `articlesDf = articlesDf.groupby("col").count()`

- 4, 5
- 2, 5, 3
- 5, 2
- 2, 3, 4
- 2, 5, 4

The code block shown below should set the number of partitions that Spark uses when shuffling data for joins or aggregations to 100. Choose the answer that correctly fills the blanks in the code block to accomplish this.spark.sql.shuffle.partitions__1__.__2__.__3__(__4__, 100) 1 point

- 1. Spark 2. Conf 3. Set 4. "spark.sql.shuffle.partitions"
- 1. Pyspark 2. Config 3. Set 4. spark.shuffle.partitions
- 1. Spark 2. Conf 3. Get 4. "spark.sql.shuffle.partitions"
- 1. Pyspark 2. Config 3. Set 4. "spark.sql.shuffle.partitions"
- 1. Spark 2. Conf 3. Set 4. "spark.sql.aggregate.partitions"

The code block displayed below contains an error. The code block should read the csv file located at path data/transactions.csv into DataFrame transactionsDf, using the first row as column header and casting the columns in the most appropriate type. Find the error. 1 point

First 3 rows of `transactions.csv`:

```
1 transactionId;storeId;productId;name
2 1;23;12;green grass
3 2;35;31;yellow sun
4 3;23;12;green grass
```

Code block:

```
transactionsDf = spark.read.load("data/transactions.csv", sep=";", format="csv", header=True)
```

- The DataFrameReader is not accessed correctly.
- The transaction is evaluated lazily, so no file will be read.
- Spark is unable to understand the file type.
- The code block is unable to capture all columns.
- The resulting DataFrame will not have the appropriate schema.

The code block shown below should read all files with the file ending .png in directory path into Spark. Choose the answer that correctly fills the blanks in the code block to accomplish this.spark.__1__.__2__(__3__).option(__4__, "*.png").__5__(path) 1 point

- 1. read()2. Format 3. "binaryFile" 4. "recursiveFileLookup" 5. load
- 1. Read 2. Format 3. "binaryFile" 4. "pathGlobFilter" 5. load
- 1. Read 2. Format 3. binaryFile 4. pathGlobFilter 5. load
- 1. Open 2. Format 3. "Image" 4. "fileType" 5. open
- 1. Open 2. As 3. "binaryFile" 4. "pathGlobFilter" 5. load

The code block displayed below contains an error. The code block should configure Spark so that DataFrames up to a size of 20 MB will be broadcast to all worker nodes when performing a join. Find the error. Code block:spark.conf.set("spark.sql.autoBroadcastJoinThreshold", 20) 1 point

- Spark will only broadcast DataFrames that are much smaller than the default value.
- The correct option to write configurations is through spark.config and not spark.conf.
- Spark will only apply the limit to threshold joins and not to other joins.
- The passed limit has the wrong variable type.
- The command is evaluated lazily and needs to be followed by an action.

The code block shown below should return a DataFrame with columns transactionId, predError, value, and f from DataFrame transactionsDf. Choose the answer that correctly fills the blanks in the code block to accomplish this.transactionsDf.__1__(__2__) 1 point

- 1. Filter 2. "transactionId", "predError", "value", "f"
- 1. Select 2. "transactionId, predError, value, f"
- 1. Select 2. ["transactionId", "predError", "value", "f"]
- 1. Where 2. col("transactionId"), col("predError"), col("value"), col("f")
- 1. Select 2. col(["transactionId", "predError", "value", "f"])

Which of the following code blocks reads all CSV files in directory filePath into a single DataFrame, with column names defined in the CSV file headers? 1 point

Content of directory `filePath`:

```
_SUCCESS
_committed_2754546451699747124
_started_2754546451699747124
part-00000-tid-2754546451699747124-10eb85bf-8d91-4dd0-b60b-2f3c02eeecaa-298-1-c000.csv.gz
part-00001-tid-2754546451699747124-10eb85bf-8d91-4dd0-b60b-2f3c02eeecaa-299-1-c000.csv.gz
part-00002-tid-2754546451699747124-10eb85bf-8d91-4dd0-b60b-2f3c02eeecaa-300-1-c000.csv.gz
part-00003-tid-2754546451699747124-10eb85bf-8d91-4dd0-b60b-2f3c02eeecaa-301-1-c000.csv.gz
```

- spark.option("header",True).csv(filePath)
- spark.read.format("csv").option("header",True).option("compression","zip").load(filePath)
- spark.read().option("header",True).load(filePath)
- spark.read.format("csv").option("header",True).load(filePath)
- spark.read.load(filePath)

The code block shown below should return a single-column DataFrame with a column named consonant_ct that, for each row, shows the number of consonants in column itemName of DataFrame itemsDf. Choose the answer that correctly fills the blanks in the code block to accomplish this. 1 point

DataFrame `itemsDf`:

```
+------+---------------------------------+-----------------------------+--------
-----------+
|itemId|itemName                         |attributes
|supplier           |
+------+---------------------------------+-----------------------------+--------
-----------+
|1     |Thick Coat for Walking in the Snow|[blue, winter, cozy]         |Sports
Company Inc.|
|2     |Elegant Outdoors Summer Dress     |[red, summer, fresh, cooling]|YetiX
            |
|3     |Outdoors Backpack                 |[green, summer, travel]      |Sports
Company Inc.|
+------+---------------------------------+-----------------------------+--------
-----------+
```

Code block:

```
itemsDf.select(__1__(__2__(__3__(__4__), "a|e|i|o|u|\s", "")).__5__("consonant_ct"))
```

- 1. Length 2. Regexp_extract 3. Upper 4. col("itemName")5. as
- 1. Size 2. Regexp_replace 3. Lower 4. "itemName" 5. alias
- 1. Lower 2. Regexp_replace 3. Length 4. "itemName" 5. alias
- 1. Length 2. Regexp_replace 3. Lower 4. col("itemName")5. alias
- 1. Size 2. Regexp_extract 3. Lower 4. col("itemName") 5. alias

Which of the following code blocks produces the following output, given DataFrame transactionsDf? 1 point

Output:

```
root
 |-- transactionId: integer (nullable = true)
 |-- predError: integer (nullable = true)
 |-- value: integer (nullable = true)
 |-- storeId: integer (nullable = true)
 |-- productId: integer (nullable = true)
 |-- f: integer (nullable = true)
```

DataFrame `transactionsDf`:

```
+-------------+---------+-----+-------+---------+----+
|transactionId|predError|value|storeId|productId|   f|
+-------------+---------+-----+-------+---------+----+
|            1|        3|    4|     25|        1|null|
|            2|        6|    7|      2|        2|null|
|            3|        3| null|     25|        3|null|
+-------------+---------+-----+-------+---------+----+
```

- transactionsDf.schema.print()
- transactionsDf.rdd.printSchema()
- transactionsDf.rdd.formatSchema()
- transactionsDf.printSchema()
- print(transactionsDf.schema)

Which of the following code blocks returns a DataFrame that is an inner join of DataFrame itemsDf and DataFrame transactionsDf, on columns itemId and productId, respectively and in which every itemId just appears once? 1 point

- itemsDf.join(transactionsDf, "itemsDf.itemId==transactionsDf.productId").distinct("itemId")
- itemsDf.join(transactionsDf, itemsDf.itemId==transactionsDf.productId).dropDuplicates(["itemId"])
- itemsDf.join(transactionsDf, itemsDf.itemId==transactionsDf.productId).dropDuplicates("itemId")
- itemsDf.join(transactionsDf, itemsDf.itemId==transactionsDf.productId, how="inner").distinct(["itemId"])
- itemsDf.join(transactionsDf, "itemsDf.itemId==transactionsDf.productId", how="inner").dropDuplicates(["itemId"])

Which of the following code blocks reads in the parquet file stored at location filePath, given that all columns in the parquet file contain only whole numbers and are stored in the most appropriate format for this kind of data? 1 point

Option 1

Option 2

Option 3

Option 4

Option 5

Which of the following code blocks generally causes a great amount of network traffic? 1 point

- DataFrame.select()
- DataFrame.coalesce()
- DataFrame.collect()
- DataFrame.rdd.map()
- DataFrame.count()

In which order should the code blocks shown below be run in order to return the number of records that are not empty in column value in the DataFrame resulting from an inner join of DataFrame transactionsDf and itemsDf on columns productId and itemId, respectively?1. .filter(~isnull(col('value')))2. .count()3. transactionsDf.join(itemsDf, col("transactionsDf.productId")==col("itemsDf.itemId"))4. transactionsDf.join(itemsDf, transactionsDf.productId==itemsDf.itemId, how='inner')5. .filter(col('value').isnotnull())6. .sum(col('value')) 1 point

- 4, 1, 2
- 3, 1, 6
- 3, 1, 2
- 3, 5, 2
- 4, 6

The code block displayed below contains an error. The code block should count the number of rows that have a predError of either 3 or 6. Find the error.Code block:transactionsDf.filter(col('predError').in([3, 6])).count() 1 point

- The number of rows cannot be determined with the count() operator.
- Instead of filter, the select method should be used.
- The method used on column predError is incorrect.
- Instead of a list, the values need to be passed as single arguments to the in operator.
- Numbers 3 and 6 need to be passed as string variables.

Which of the following code blocks returns a new DataFrame with the same columns as DataFrame transactionsDf, except for columns predError and value which should be removed? 1 point

- transactionsDf.drop(["predError", "value"])
- transactionsDf.drop("predError", "value")
- transactionsDf.drop(col("predError"), col("value"))
- transactionsDf.drop(predError, value)
- transactionsDf.drop("predError & value")

In which order should the code blocks shown below be run in order to read a JSON file from location jsonPath into a DataFrame and return only the rows that do not have value 3 in column productId?1. importedDf.createOrReplaceTempView("importedDf")2. spark.sql("SELECT * FROM importedDf WHERE productId != 3")3. spark.sql("FILTER * FROM importedDf WHERE productId != 3") 4. importedDf = spark.read.option("format", "json").path(jsonPath)5. importedDf = spark.read.json(jsonPath) 1 point

- ◯ 4, 1, 2
- ◯ 5, 1, 3
- ◯ 5, 2
- ◯ 4, 1, 3
- ◯ 5, 1, 2

The code block displayed below contains an error. The code block should return a DataFrame in which column predErrorAdded contains the results of Python function add_2_if_geq_3 as applied to numeric and nullable column predError in DataFrame transactionsDf. Find the error. 1 point

Code block:

```
def add_2_if_geq_3(x):
  if x is None:
    return x
  elif x >= 3:
    return x+2
  return x

add_2_if_geq_3_udf = udf(add_2_if_geq_3)

transactionsDf.withColumnRenamed("predErrorAdded",
add_2_if_geq_3_udf(col("predError")))
```

- ◯ The operator used to adding the column does not add column predErrorAdded to the DataFrame.
- ◯ Instead of col("predError"), the actual DataFrame with the column needs to be passed, like so transactionsDf.predError.
- ◯ The udf() method does not declare a return type.
- ◯ UDFs are only available through the SQL API, but not in the Python API as shown in the code block.
- ◯ The Python function is unable to handle null values, resulting in the code block crashing on execution.

Submit Clear form